API Aggregation Service Documentation

Georgios Fotopoulos

Overview

The **API Aggregation Service** provides a unified interface to access aggregated data from multiple external APIs: news, weather, and Pokémon information. It includes error handling and fallback mechanisms to ensure reliable responses even if an external API becomes temporarily unavailable.

Prerequisites

- **.NET 8**
- **Visual Studio** (or any compatible IDE)
- **API Keys**:
    - **News API Key**: Required for accessing the News API, obtainable from NewsAPI at https://newsapi.org/.
    - **OpenWeatherMap API Key**: Required for accessing the Weather API, obtainable from OpenWeatherMap at https://openweathermap.org/.
    - **PokéAPI**: No API key required, accessible at https://pokeapi.co/.

Setup Instructions

1. **Clone the Project**: Clone or download the repository to your local machine.
2. **Add API Keys**: Currently, the API keys are hardcoded in the services for demonstration. Ensure to replace these hardcoded keys in the NewsService and WeatherService classes with your own API keys:
    - NewsService.cs: Replace "140cd018eaf64dc9b16210dab3ca1fda" with your News API key.
    - WeatherService.cs: Replace "9a0b8e1571e5ab2e7049843b93648ac0" with your OpenWeatherMap API key.

Endpoints

1. News Endpoint

- **URL**: /api/aggregation/news
- **Method**: GET
- **Description**: Fetches news articles based on a specified keyword. Supports sorting and filtering.
- **Parameters**:
    - keyword (string, required): The search term for finding news articles.
    - sortBy (string, optional): Sorting options, such as by publishedAt or relevance.
    - filterBy (string, optional): Additional filter criteria.
- **Example Request**: GET /api/aggregation/news?keyword=technology&sortBy=publishedAt
- **Example Response**:
    - title: "Sample News Article"
    - description: "Sample Description"
    - url: "https://sample.news/article"
    - publishedAt: "2024-10-15T10:00:00Z"

2. Weather Endpoint

- **URL**: /api/aggregation/weather
- **Method**: GET
- **Description**: Retrieves weather information for a specified location, with optional sorting and filtering.
- **Parameters**:
    - location (string, required): City name for which weather data is requested.
    - sortBy (string, optional): Options like sorting by temperature or city.
    - filterBy (string, optional): Filter based on weather conditions, such as clear.
- **Example Request**: GET /api/aggregation/weather?location=London&sortBy=temperature
- **Example Response**:
    - city: "London"
    - temperature: 15
    - condition: "Partly cloudy"

3. Pokémon Endpoint

- **URL**: /api/aggregation/pokemon
- **Method**: GET
- **Description**: Retrieves a list of Pokémon based on their primary or secondary type, with sorting options.
- **Parameters**:
    - primaryType (string, required): Main type of Pokémon (e.g., fire).
    - secondaryType (string, optional): Secondary type for dual-type Pokémon.
    - sortBy (string, optional): Sorting options like name or id.
- **Example Request**: GET /api/aggregation/pokemon?primaryType=fire&secondaryType=flying&sortBy=name
- **Example Response**:
    - id: 6
    - name: "Charizard"
    - types: ["fire", "flying"]
    - height: 17
    - weight: 905

Error Handling and Fallback

Each service includes error handling and fallback mechanisms:

- **News Service**: If the News API is unavailable, an empty list is returned, with a console message logged for troubleshooting.
- **Weather Service**: If the Weather API is unavailable, a default response with Temperature = 0 and Condition = "Unavailable" is returned.
- **Pokémon Service**: If the PokéAPI is unavailable, an empty list is returned, along with an error message logged to the console.

Testing

The project includes unit tests for each service, which are located in the ApiAggregatorService.Tests project. These tests cover:

- **Successful API responses** for each service.

- **Error handling and fallback responses** to verify the resilience of each service.
- **Sorting and filtering functionality** to ensure results are processed as expected.

Running Tests

1. Open the ApiAggregatorService.Tests project in your IDE.
2. Run all tests to verify the functionality and resilience of each service.

Notes

- **Error Logs**: Errors are currently logged to the console for simplicity. For production use, consider implementing a more robust logging framework.